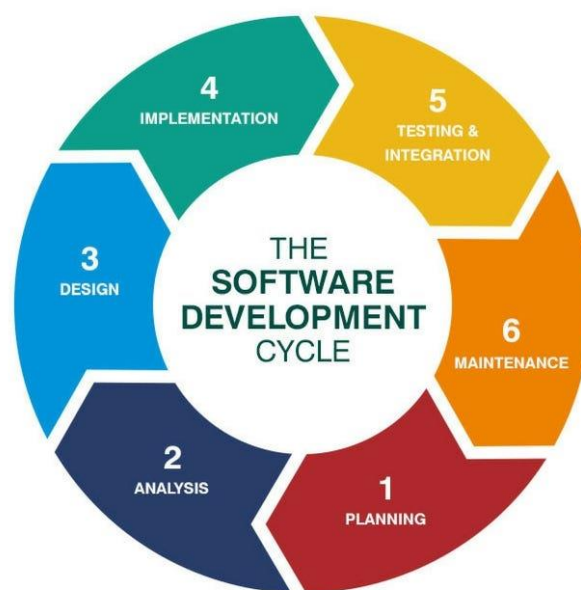


Introduction

This document offers an in-depth examination of Echo Gallery's creation, a Single-Page Web Application (SPA) for image sharing with social media features, developed as the project for the COMP1004 module. It charts the entire project lifecycle from conception through Software Development Life Cycle (SDLC) methods, detailing the project's groundwork, design evolution, technology choices, and the obstacles overcome. The report highlights developmental successes and the agile Scrum framework's key role in planning and refinement, with sprints and GitHub commits, demonstrating the project's growth.

Software Development Life Cycle

The 6 stages of the Agile SDLC are as follows:



Synotive

Agile SDLC Diagram (Pineiro, 2018)

The SDLC is a benchmark for the industry, with each stage being as crucial as the one before. It offers a structured path for software development, critical for a well-coordinated process, as highlighted by Development (2023). Skipping stages and diving into development without this framework risks improper execution and a final product riddled with errors. Adherence to the SDLC is not merely about following best practices; it's about guaranteeing that the product endures market pressures and fulfils user expectations.

In the next parts of this report, I will explain about each step of the Agile SDLC used, highlighting the indispensable value and specific contributions of every phase used. This report will explore how the journey from a simple idea to a fully functioning software involves distinct steps that contribute to enhancing the software's quality and effectiveness. This thorough analysis aims to provide a clear understanding of how a methodical approach to the SDLC can lead to the creation of a robust, efficient, and successful software product.

Project Vision

My vision is to develop a secure, engaging SPA that is compliant with legal standards, prioritising user needs and safety. The platform will support vibrant community interaction, where users can upload and share images freely while adhering to copyright laws, GDPR, data security protocols, and accessibility guidelines.

Compliance and User Safety:

- I'm dedicated to a secure experience, observing data protection laws, complying with GDPR (2018), and integrating privacy by design. The platform will maintain the highest security standards and respect intellectual property through educational tools and content moderation.

Performance:

- Utilising modern frontend technologies, the SPA will offer fast loading times and smooth transitions, essential for user engagement on various devices.

Data Security:

- Data security is paramount, incorporating advanced encryption, secure coding, and regular audits to safeguard against cyber threats and unauthorised access.

Accessibility:

- The platform will be accessible to all, complying with WCAG standards (W3C, 2018) to ensure it is easy for everyone to navigate, understand, and interact with.

Community and Interaction:

- Central to the platform, I aim to foster interaction through comments, reactions, and personalised galleries. Social interaction and custom content feeds will enhance the social environment.

This platform is not just a tool but a destination for creativity, cultural exchange, and community, promoting user rights, active participation, and accessibility for all.

Background

Social media is increasingly becoming more popular with the rise of technology through phones and ease of access. As Facebook has over 2.3 billion users and other social media sites being used by more than two-thirds of internet users and is still growing (Ortiz-Ospina, 2019). This shows that there is a growing demand for users to exchange everyday experiences, photographs, and establish connections with one another (University of Cumbria, 2019). Echo Gallery enters this space as a fresh social media site focused on image sharing, promoting community interaction through likes and comments.

Plan

The planning stage is vital in the SDLC, laying the groundwork for the project. It outlines the project goals, delineates the scope, assesses potential risks, and allocates resources and timelines. Effective planning provides a clear set of objectives and a strategic approach to attain them. Neglecting this stage can result in wasted resources and project setbacks.

Below is a sprint plan that reflects a methodical approach to development, prioritising the user interface (UI) and progressing logically from simple layouts to advanced features like commenting and image uploads. Time is set aside for user feedback and thorough testing, ensuring the product is refined and meets user needs. The concluding weeks focus on project review and presentation preparation, providing a chance to reflect on successes and obstacles.

Sprint Plan

Week	From	Description
1	22 Jan	Analysis – research information/photo upload based websites (e.g. Pinterest, Instagram), create User Stories, UML Diagrams and Use Case to aid with development
2	29 Jan	Design – Final design choices of UI. Incorporate insights from analysis phase into practise
3	5 Feb	Sprint 1 – Pre-loaded images displayed in the UI and main body layout
5	12 Feb	Sprint 2 – Develop navigation bar
6	19 Feb	Sprint 3 – Upload image function and box
9	26 Feb	Sprint 4 – Implement when user uploads photo with comment/information and able to add name and theme
9	4 March	Sprint 5 – Implement when user uploads image it is displayed in main body with all details entered
10	11 March	Sprint 6 – Create a function that when clicked, it shows more information about the image instead of just image and text, giving a collage feel.
11	18 March	Sprint 7 – Add comment and like/dislike functionality
12	25 March	Sprint 8 – Finalise any GUI development. Get feedback from users to note any bugs or improvements that can be made
13	1 April	Sprint 9 – Final testing and resolve any bugs that may come up
14	8 April	Review and finalise project report
15	15 April	Final due diligence. Ensure everything is in order before the portfolio submission.
Portfolio due Thursday 18th of April		
16	22 April	Revise and make notes of what went well along the way and what was difficult for presentation
17	29 April	Prepare for presentation
18	6 May	Presentation
Note:		Report will be updated when progress is made every week Unit testing will be done throughout the project

Backlog (Actual Work Accomplished)

From	Description
Jan 21 - Jan 27, 2024	<ul style="list-style-type: none"> - Added navigation bar and upload image button to begin the project interface.
Feb 12 - Feb 18, 2024	<ul style="list-style-type: none"> - Focused on core interface features, including adding images and buttons to the main layout
Mar 5 - Mar 11, 2024	<ul style="list-style-type: none"> - Developed the image upload button functionality, ensuring users can upload images. - Created firebase-init.js for Firebase initialisation - Further modified navbar - Moved the JS Code for clarity and tried to get images from Firebase to be shown
Mar 12 - Mar 18, 2024	<ul style="list-style-type: none"> - Successfully retrieved images from the database and displayed them on the page with associated content - Upload function modified to capture more fields - When user uploads photo is it updated in main body functionality - Open image context modal functionality created so context for an image is dynamically displayed - Implemented like and dislike button functionalities and fixed related bugs - Added comment section - Fixed bug with close button - Fixed upload button CSS - Created Sign Up, Log In, Sign Out button and functionality
Mar 19 - Mar 25, 2024	<ul style="list-style-type: none"> - Enhanced Open Image Context Modal function - Modified, and enhanced the comment functionality - Further fixed like and dislike button function - Tackled navbar functionality and added confirmation step for password setting - Updated spacing in HTML and fixed issues with navigation bar and scrolling functions - Updated CSS for like and dislike button - Added password confirmation
Mar 26 - Apr 1, 2024	<ul style="list-style-type: none"> - Added delete function - Updated code to allow users to edit image context fields and improved word count error handling - Fixed like and dislike button bugs - Updated CSS for the edit and delete button
Apr 2 - Apr 8, 2024	<ul style="list-style-type: none"> - Fixed a crucial bug where reactions subcollection was not removed, ensuring data integrity. - Final testing completed
Apr 9 - Apr 15, 2024	<ul style="list-style-type: none"> - Added feedback forms and made minor adjustments to code for better functionality and user engagement - Updated comments for clarity - Finished project poster and finalised all documentation

Requirements Analysis

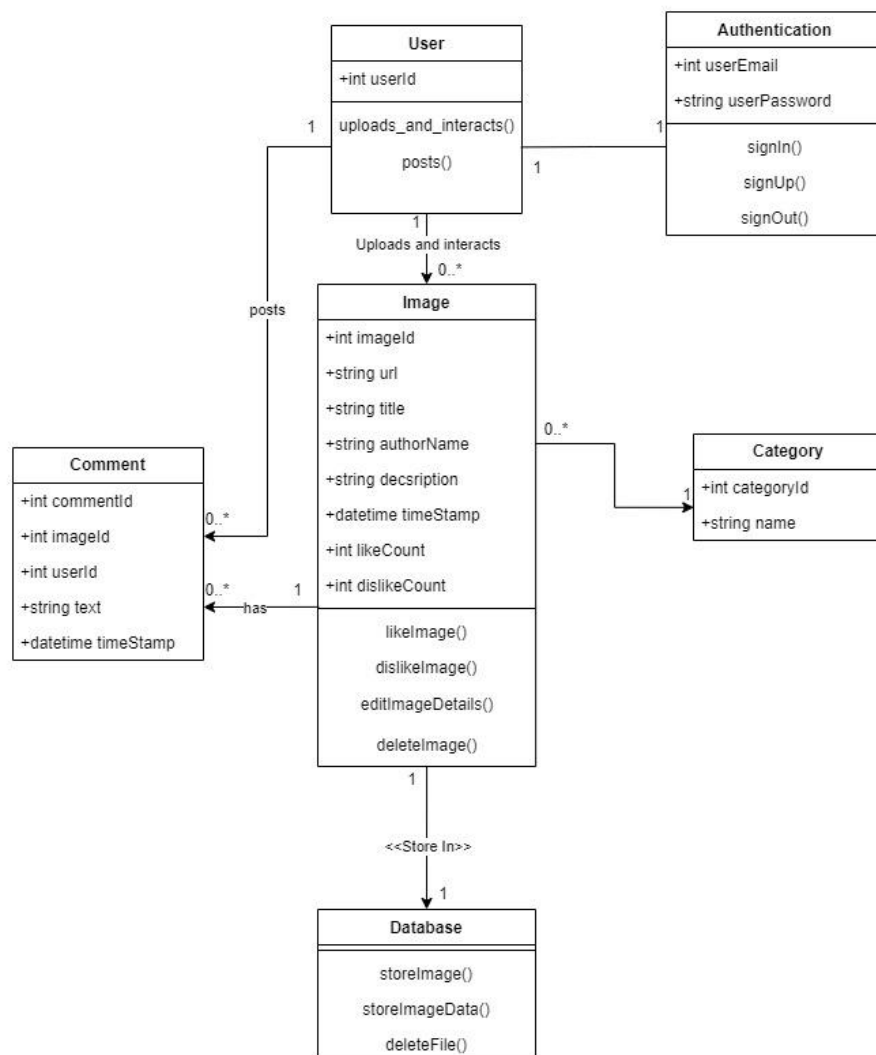
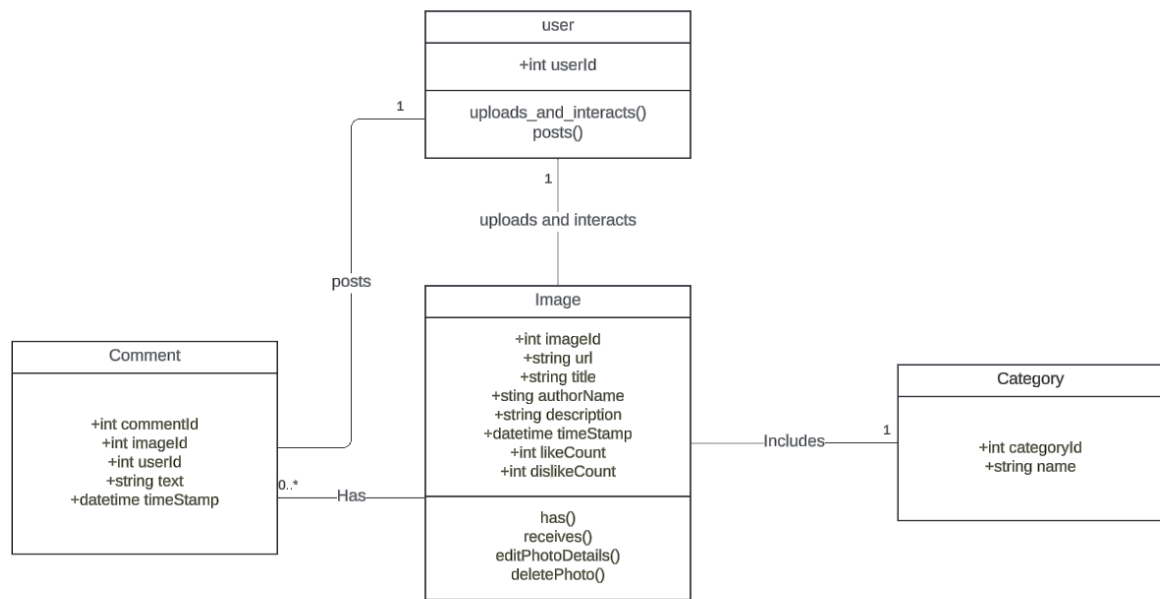
The Requirements Analysis stage is a crucial phase in the SDLC, as it establishes the project's foundation. It identifies and documents software functions and performance, guiding subsequent design and implementation. Effective analysis ensures proper project scoping, prevents feature creep, and minimises the risk of failing to meet user needs. This documentation provides clear guidelines for later SDLC stages, clarifying development objectives.

Below are diagrams illustrating my requirements analysis.

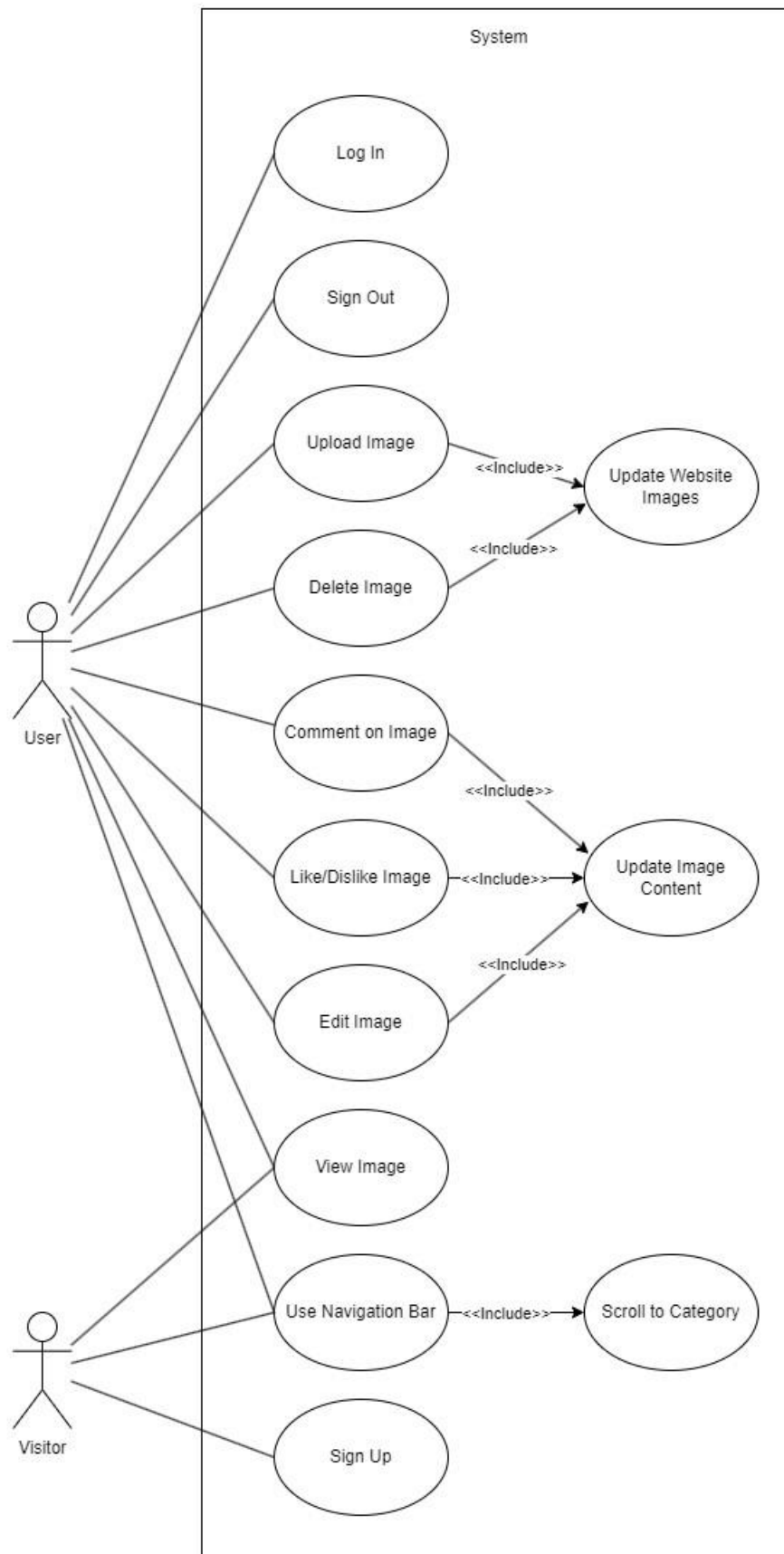
User stories

User	Visitor
As a user or visitor, I want to view images content.	
As a user or visitor, I want a navigation bar so that I can easily access different sections of the site.	
As a user, I want to log in.	As a visitor, I want to sign up so that I can create my own account, upload images, and interact with other users.
As a user, I want to securely sign out.	
As a user, I want to comment on images.	
As a user, I want to like or dislike images.	
As a user, I want to upload images.	
As a user, I want to edit my uploaded images.	
As a user, I want to delete images that I have uploaded if I no longer want them to be displayed on the platform.	
As a user, I want to delete my uploaded images.	

UML Diagram



Use Case



Use Case Description

Name	View Image
Short Description	Allows users to view details of an image, including a larger version, comments, and likes/dislikes.
Precondition	The image is publicly available, or the user has permission to view it. The system is operational
Post Condition	User clicks on the arrow on the image, views the image and available interactions (comments, likes, dislikes etc).
Error Situations	Image fails to load due to a broken link or server error.
System state in the event of an error	The system displays an error message and does not show the image.
Actors	User, Visitor
Triggers	User selects an image to view from the gallery.
Standard Process	<ol style="list-style-type: none">1. User clicks on an image thumbnail.2. System retrieves and displays the full-size image along with uploader, upload date, comments, likes, and dislikes.3. User can read comments and view like/dislike counts.
Alternative Process	<ol style="list-style-type: none">4' If the image cannot be displayed, offer an error message and possibly a thumbnail or placeholder image.

Name	Upload Image
Short Description	Upload an image
Precondition	User is logged in. The system is operational
Post Condition	The image is displayed in the gallery with the provided details
Error Situations	The image file format is not supported. The image file is too large. There's a network or server error preventing upload.
System state in the event of an error	The system remains stable. The image is not uploaded, and an error message is displayed.
Actors	User
Triggers	The user selects upload image
Standard Process	<ol style="list-style-type: none"> 1. User clicks on the 'Upload Image' button. 2. User selects an image file to upload. 3. Image Preview is displayed. 4. User enters image details (title, description). 5. System validates the image file and details. 6. System uploads the image, updates the gallery, and confirms the upload to the user.
Alternative Process	<ol style="list-style-type: none"> 7' If the image file is not in a supported format or exceeds size limits, the system prompts the user to select a different file. 8' If there's a network or server error, the system offers the user the option to retry the upload.

Name	Edit Image
Short Description	Allows a user to edit the details of an image they have previously uploaded
Precondition	User is logged in. Must be owner of the image. The system is operational
Post Condition	The image details are updated in the gallery.
Error Situations	User tries to edit an image they do not own. Invalid input for image details. Network or server error during the update process
System state in the event of an error	The system displays an error message and does not apply the changes.
Actors	User
Triggers	The user selects the 'Edit Image' option for one of their images.
Standard Process	<ol style="list-style-type: none"> 1. User selects an image from their gallery and chooses the 'Edit' option. 2. User modifies the image details such as title, description, etc. 3. System validates the new input details. 4. System updates the image details and confirms the changes to the user.
Alternative Process	<ol style="list-style-type: none"> 5' If the user does not own the image, the edit option is not available. 6' If there is invalid input, the system prompts for correction before submission. 7' If a network or server error occurs, the system provides an option to retry.

Name	Delete Image
Short Description	Allows a user to remove an image they have uploaded from the gallery.
Precondition	User is logged in. Must be owner of the image. The system is operational
Post Condition	The image is deleted from the gallery and is no longer accessible.
Error Situations	The system remains stable. Network or server error during the deletion process.
System state in the event of an error	The system displays an error message and does not apply the changes. The image is not deleted, and an error message is displayed.
Actors	User
Triggers	The user selects the 'Delete Image' option for one of their images.
Standard Process	<ol style="list-style-type: none"> 1. User selects an image from their gallery and chooses the 'Delete' option. 2. The system asks the user to confirm the deletion. 3. Upon confirmation, the system deletes the image. 4. System updates the gallery and confirms the deletion to the user.
Alternative Process	<ol style="list-style-type: none"> 5' If the user does not own the image, the delete option is not available. 6' If a network or server error occurs, the system provides an option to retry the deletion.

Name	Sign Up
Short Description	Allows visitors to create a new user account in the system.
Precondition	Visitor is not already logged in.
Post Condition	A new user account is created, and the user is logged in.
Error Situations	Invalid input (e.g., non-valid email format). Email already registered. Server or database error during account creation.
System state in the event of an error	No new account is created, and the visitor is prompted to correct the input or try again later.
Actors	Visitor
Triggers	Visitor selects the 'Sign Up' option.
Standard Process	<ol style="list-style-type: none"> 1. Visitor fills out the sign-up form with email, username, and password. 2. System validates the provided details. 3. System creates a new user account and logs the user in. 4. System confirms account creation to the user.
Alternative Process	<ol style="list-style-type: none"> 5' If the email is already in use, prompt to try a different email. 6' If the input is invalid, prompt to correct the details.

Name	Log In
Short Description	Allows users to access their account by providing credentials.
Precondition	User is registered and not currently logged in.
Post Condition	User is granted access to their account.
Error Situations	Incorrect credentials. User account is locked or disabled. Server or database error during authentication.
System state in the event of an error	Access is not granted, and the user is informed of the error.
Actors	User
Triggers	User selects 'Log In' and submits credentials.
Standard Process	<ol style="list-style-type: none"> 1. User enters their email and password. 2. System validates the credentials. 3. System grants access and transitions to the user's profile or homepage.
Alternative Process	<ol style="list-style-type: none"> 4' If credentials are incorrect, deny access and prompt to try again or reset password.

Name	Sign Out
Short Description	Allows users to securely exit their accounts.
Precondition	User is currently logged in.
Post Condition	User session is terminated, and the user is redirected to the homepage or sign-in page.
Error Situations	<p>User's session does not close properly due to a server error.</p> <p>There is a network error preventing the sign-out request from reaching the server.</p>
System state in the event of an error	The system maintains the user's session and prompts the error, advising to retry.
Actors	User
Triggers	User clicks the 'Sign Out' button.
Standard Process	<ol style="list-style-type: none"> 1. The user clicks the 'Sign Out' button in the application interface. 2. The system processes the sign-out request and terminates the user's session. 3. The user is redirected to the homepage or sign-in page, and a message confirming successful sign-out is displayed.
Alternative Process	<ol style="list-style-type: none"> 4' If the sign-out process fails, inform the user, and provide the option to attempt to sign out again.

Name	Comment on Image
Short Description	Allows users to add comments to images in the gallery.
Precondition	User must be logged in.
Post Condition	Comment is publicly visible on the image's context page.
Error Situations	<p>Comment submission fails due to a network or server error.</p> <p>Comment contains prohibited content, triggering moderation.</p>
System state in the event of an error	Comment is not posted, and the user is notified of the failure.
Actors	User
Triggers	User submits a comment on an image's page by selecting 'Post Comment' or pressing enter.
Standard Process	<ol style="list-style-type: none"> 1. User types a comment in the comment field. 2. System validates the comment. 3. System posts the comment below the image. 4. System displays the comment to all viewers of the image.
Alternative Process	<ol style="list-style-type: none"> 5' If there's a network error, provide an option to retry. 6' If the comment is above the word max, provide an error with how many they have used and what is the max

Name	Like/Dislike Image
Short Description	Allows users to express their opinion on an image with a like or dislike.
Precondition	User must be logged in.
Post Condition	Like or dislike count is updated, and user's preference is recorded.
Error Situations	Like/dislike action fails due to network or server error.
System state in the event of an error	The like or dislike is not registered, and the user is prompted to try again.
Actors	User
Triggers	User clicks the 'Like' or 'Dislike' button on an image.
Standard Process	<ol style="list-style-type: none"> 1. User selects the 'Like' or 'Dislike' button. 2. System updates the like or dislike count for the image. 3. System records the user's action to prevent multiple votes.
Alternative Process	<ol style="list-style-type: none"> 4' If there's a network error, provide an option to retry.

Name	Use Navigation Bar
Short Description	Allows users to navigate through different sections of the website using the navigation bar.
Precondition	The website is accessible, and the navigation bar is visible.
Post Condition	User is taken to the selected section of the website.
Error Situations	The selected section does not load due to a network or server error. The user attempts to navigate to a restricted area without appropriate permissions.
System state in the event of an error	The system remains on the current section, and an error message is displayed.
Actors	User, Visitor
Triggers	User selects an option in the navigation bar.
Standard Process	<ol style="list-style-type: none"> 1. User clicks on a section title in the navigation bar. 2. System loads and displays the selected section. 3. System updates the URL and browser history for possible future navigation.
Alternative Process	<ol style="list-style-type: none"> 4' If a section fails to scroll, display an error, and provide a retry option.

Functional Requirements

1. The platform shall allow users to create an account using an email address and a password.
2. The platform shall authenticate users via email and password upon logging in.
3. The platform shall enable users to upload images along with titles, author name and descriptions.
4. The platform shall allow users to log out.
5. Visitors without an account and users shall have the ability to view images.
6. The platform shall have a database to read from and write to.
7. The platform shall enforce privacy settings to ensure user-uploaded images are only editable or can be deleted by the uploader.
8. The platform shall ensure all user data is handled in compliance with GDPR.
9. The platform shall provide error messages and guidance for users when interactions fail, or inputs are invalid.
10. The platform shall include a navigation bar for easy access to different sections.
11. The platform shall display images in an organised gallery view.
12. All user interactions like sign-up, log in, image upload, and comments shall be logged for potential auditing.
13. The system shall be maintainable with clear documentation for future development and updates.
14. Users shall have the ability to like or dislike images.
15. The platform shall provide a comment feature on images for logged-in users.
16. The platform shall allow users to edit the details of images they have uploaded.
17. The platform shall permit users to delete their uploaded images.
18. The platform shall be responsive and functional on various devices and screen sizes.

Must: 1-9

Should: 10-13

Could: 14-18

Non-functional Requirements

1. The platform's user interface shall be intuitive and user-friendly, requiring minimal instruction for new users to navigate.
2. The platform shall ensure data security, especially user emails and authentication data.
3. The platform shall be developed with scalability in mind, able to accommodate an increasing number of users and images.

Design

The design phase is vital in the SDLC, acting as a blueprint for the project. It transforms software specifications into a design plan, encompassing architectural design, component selection, UI design, and definitions of data structures, algorithms, and detailed software architecture. This early focus ensures scalability, maintainability, and compliance with requirements, identifying potential issues to streamline development. Effective design reduces development time and ensures the final product meets user expectations and quality standards.

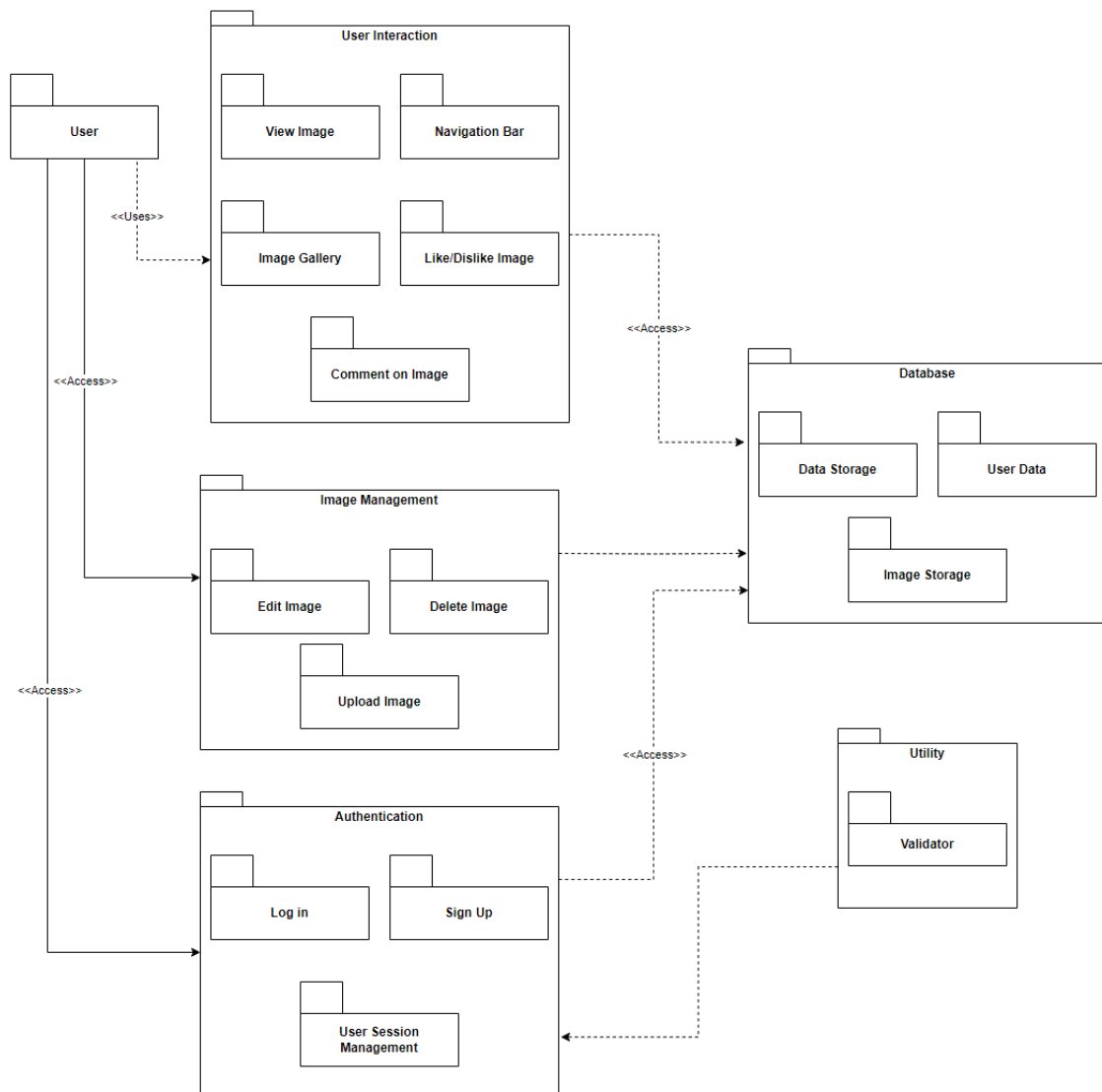
Architecture

The development of this SPA for an image sharing social media platform is a complete project that brings together an array of technologies and services for good user experience. At its core, the platform features image management capabilities, enabling users to upload, view, and interact with visual content in a dynamic, intuitive interface.

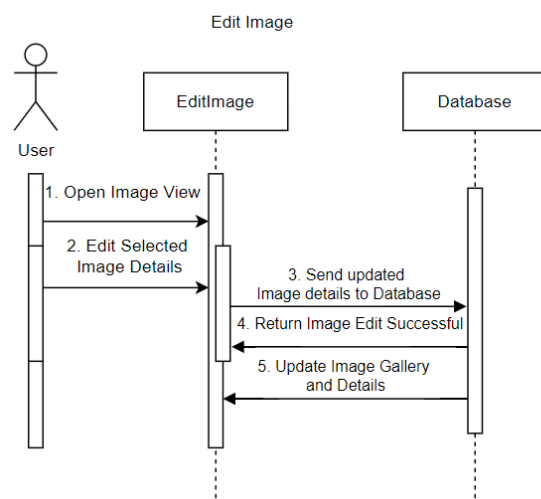
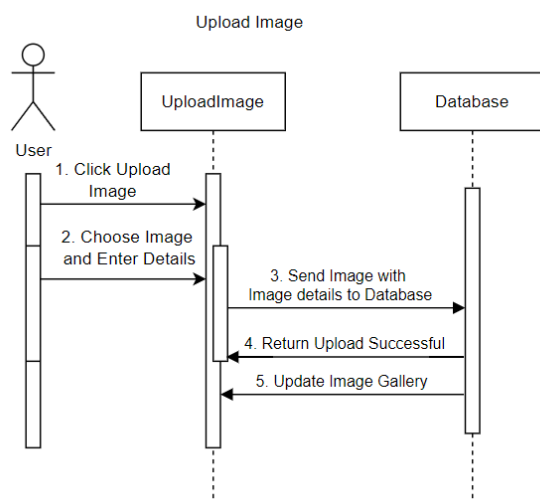
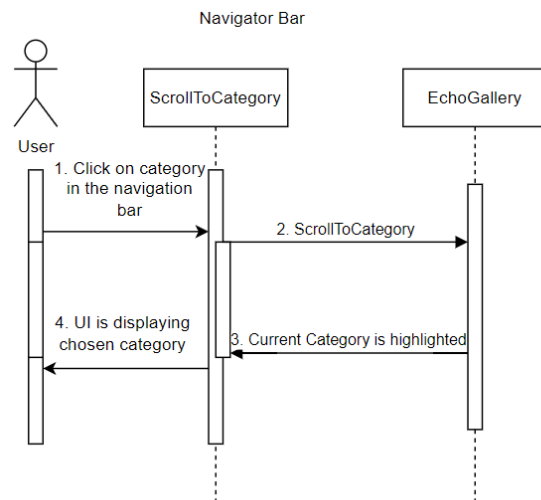
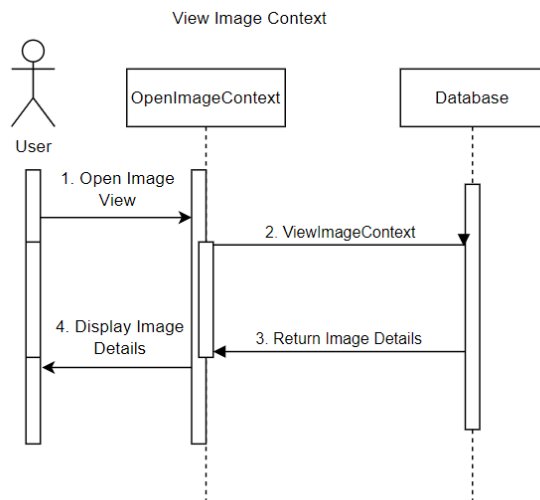
It incorporates sophisticated user authentication mechanisms, courtesy of Firebase, ensuring secure access and personalised sessions. The application also taps into the cloud-based Firestore database for efficient, real-time storage and retrieval of images and user-generated metadata. The streamlined architecture not only facilitates image categorisation and community engagement through likes and comments but also maintains a SPA structure for smooth user navigation.

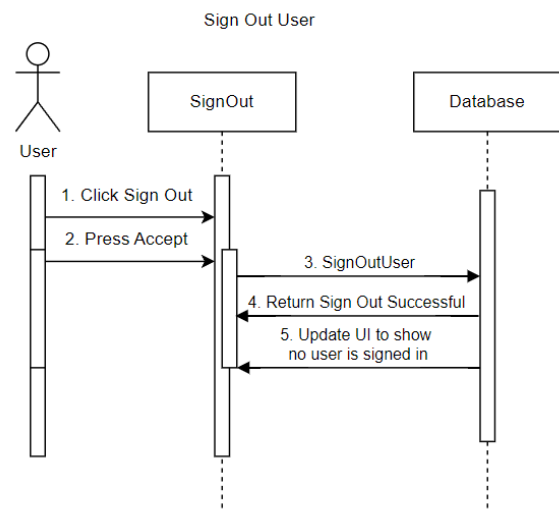
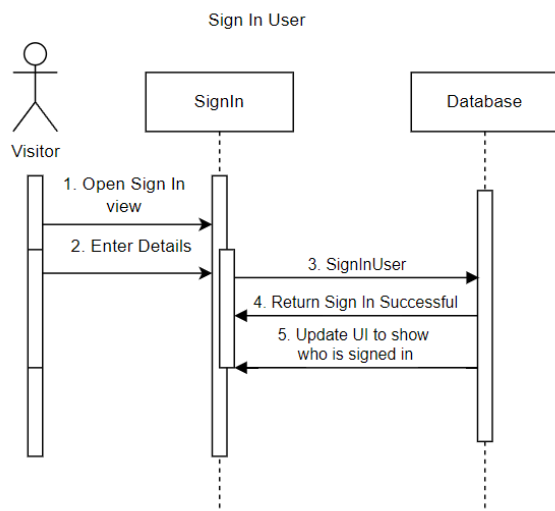
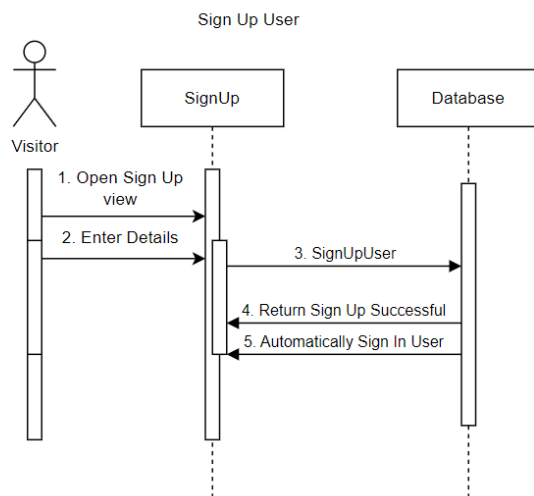
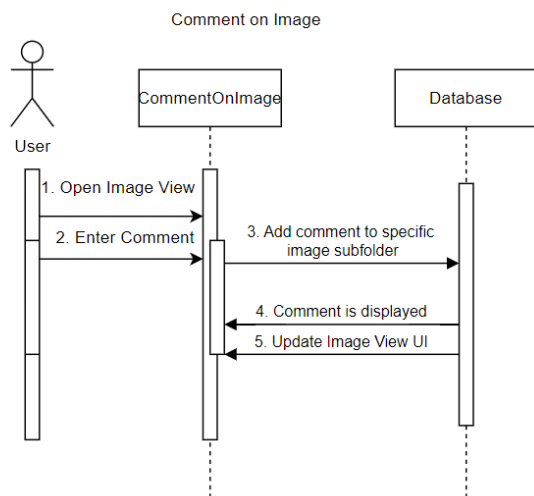
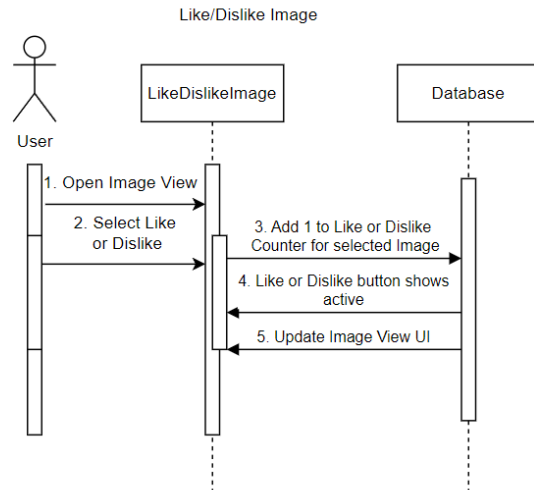
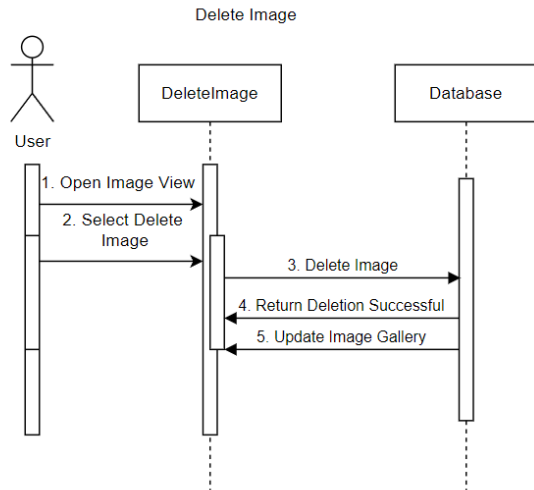
Below are diagrams illustrating my design phase.

Package Diagram

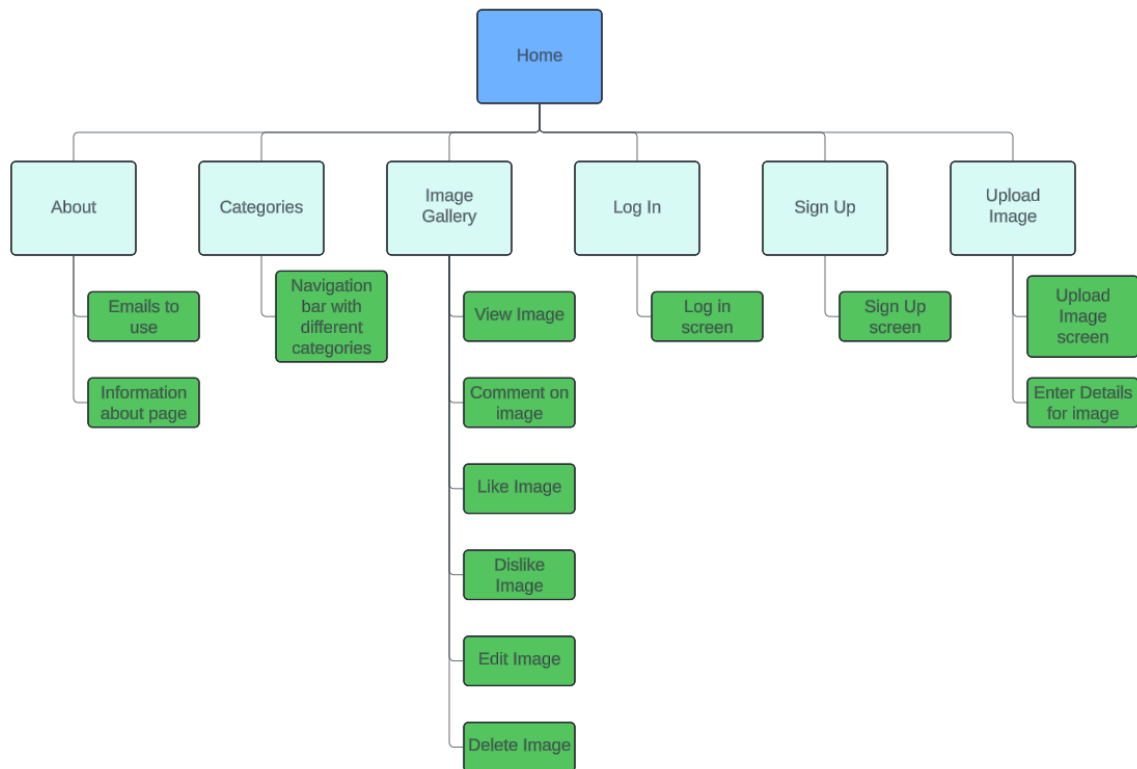


Sequence Diagram

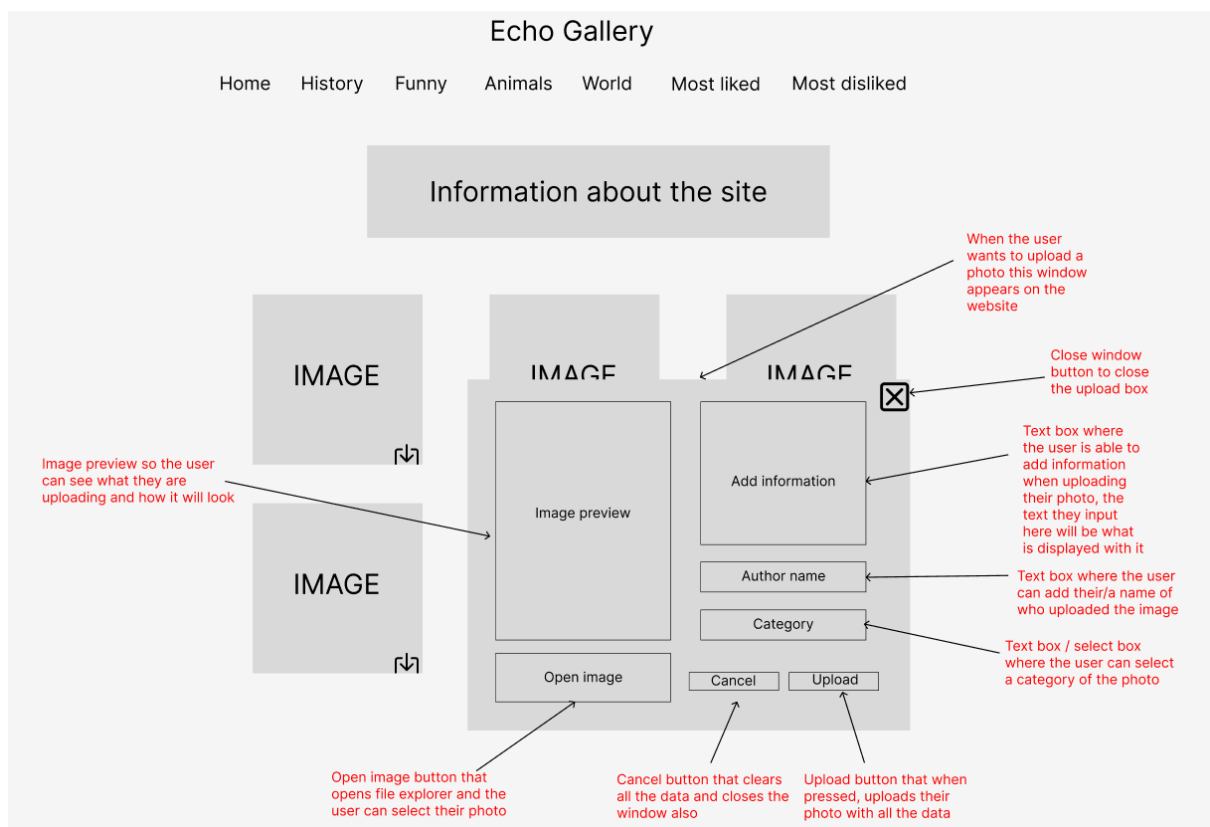
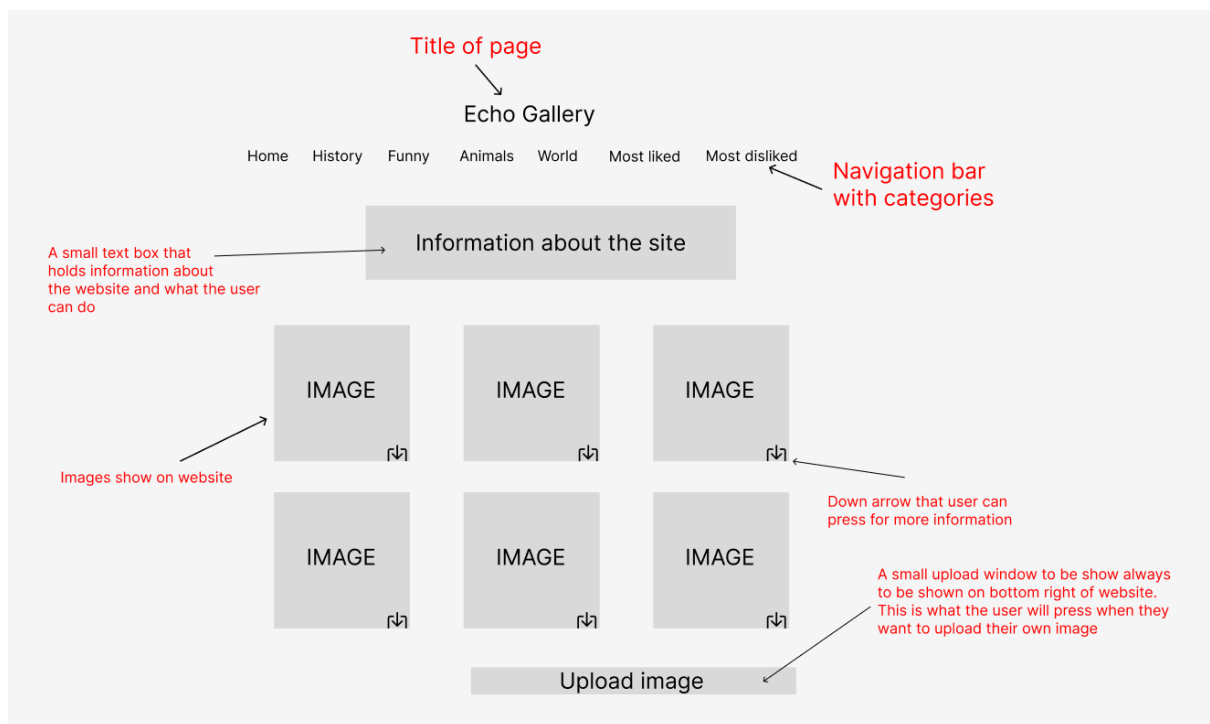


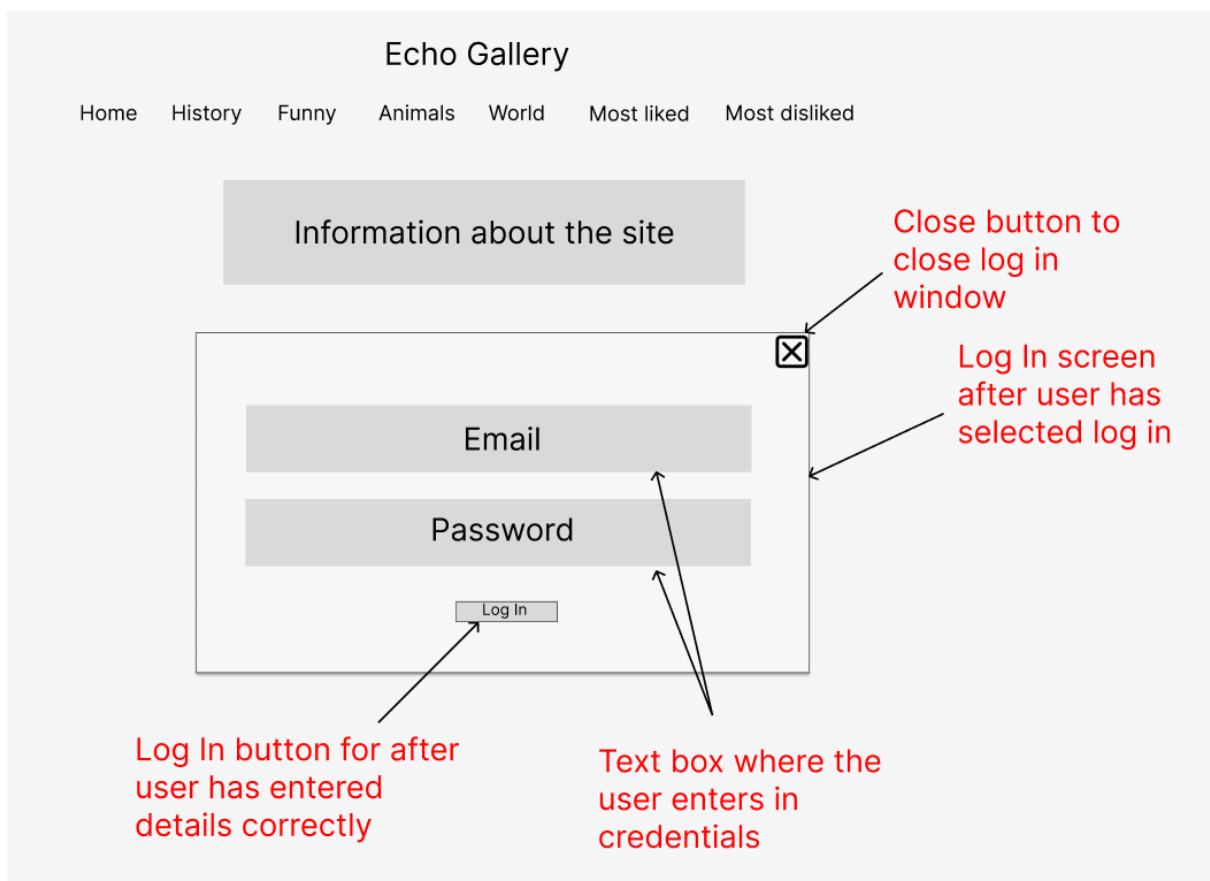
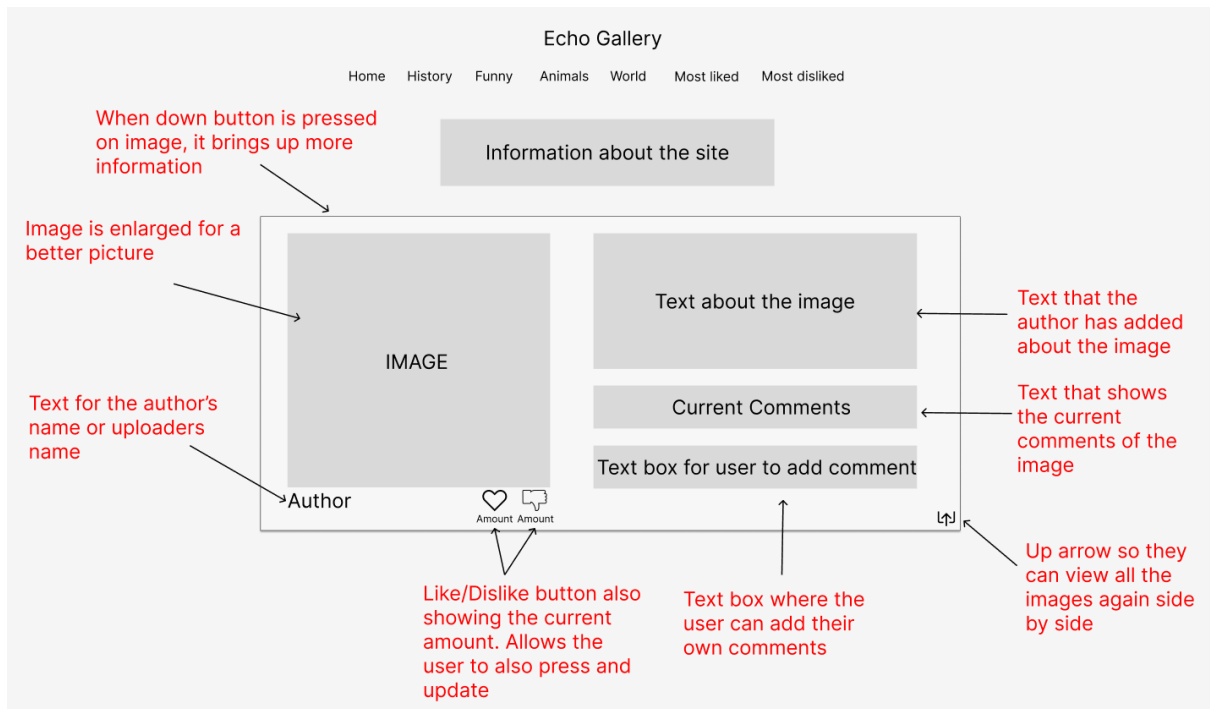


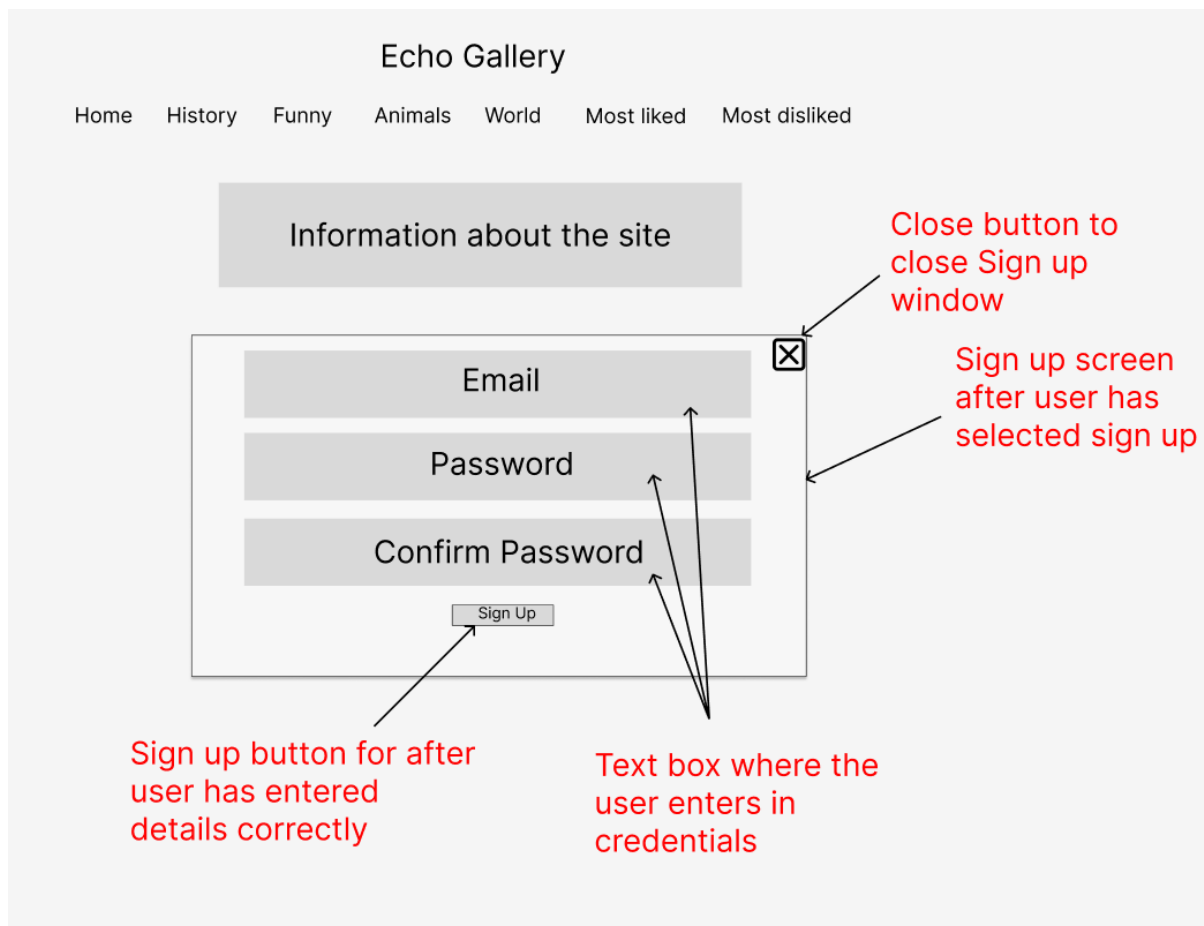
Sitemap



Wireframes







Implementation

The Implementation stage is where coding and software construction take place. Following detailed planning and design, this phase brings the project to life. Here, design documents and specifications are converted into source code, incorporating necessary components and libraries. Characterised by coding, unit testing, and integrating modules into a cohesive system, this stage marks the actual development of the software.

Below is review of the sprints carried out in my project. It details the struggles and achievements encountered throughout the development process.

Review of Sprints

Sprint 1 – This was a difficult sprint to start with. At first, I decided to have an “images” folder and manually have the sources of the images embedded into the HTML, however I decided against this. Instead, I used a database (Firebase, Firestore) which allowed me to store the images (Figure 1) into a database and dynamically displayed through JavaScript by looping through and searching for categories within the database (Figure 2, 3, 4). This approach significantly streamlined the process for managing image-related data, such as likes, dislikes, comments, and categories in subsequent sprints, leveraging Firebase's data handling capabilities.

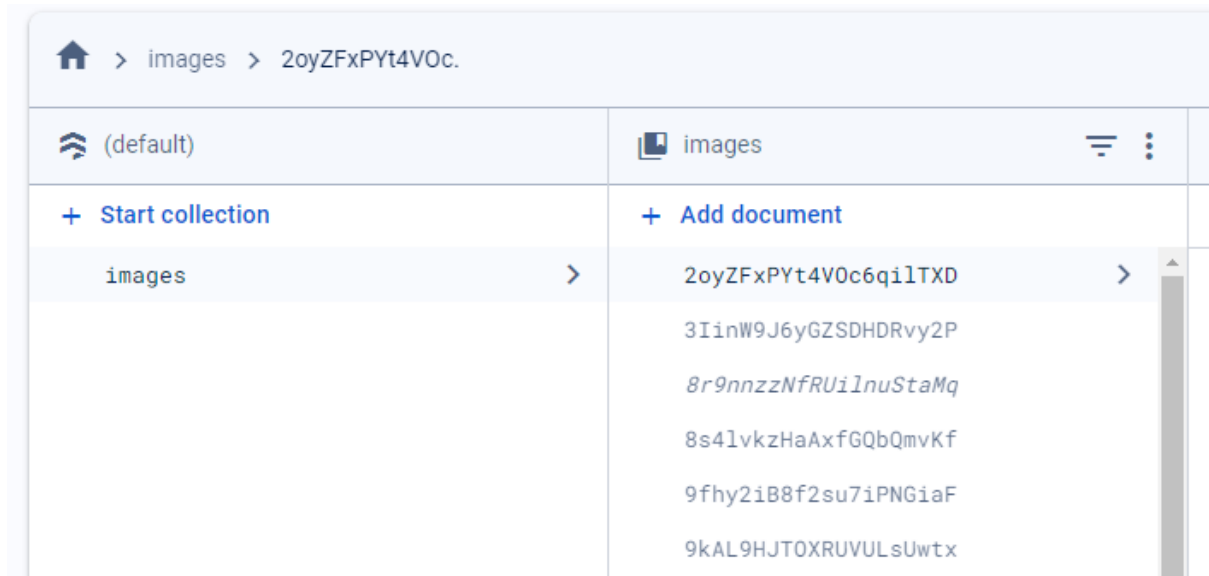


Figure 1

```
// Display images for each category
const categories = ['food', 'fashion', 'sports', 'informative', 'funny', 'history', 'other'];
categories.forEach(async (category) => {
  await displayImagesByCategory(category);
  highlightNavButton();
});
```

Figure 2

```
// Function to display images by category
async function displayImagesByCategory(categoryName) {
  // Reference to the container in your HTML where the images will be displayed
  console.log(`Querying for .${categoryName}-images .image-gallery`);
  const categoryContainer = document.querySelector(`.${categoryName}-images`);

  // Check if the container exists
  if (!categoryContainer) {
    console.error(`The container for category "${categoryName}" does not exist.`);
    return;
  }

  // Clear out any existing content in the container
  categoryContainer.innerHTML = '';

  // Query Firestore for images in the specified category
  const imagesCollectionRef = collection(db, 'images');
  const q = query(imagesCollectionRef, where("category", "==", categoryName));
  const querySnapshot = await getDocs(q);

  // Loop through the documents returned by the query
  for (const docSnapshot of querySnapshot.docs) {
    const data = docSnapshot.data();
    const imageGridItem = document.createElement('div');
    imageGridItem.className = 'image-grid-item';
    imageGridItem.setAttribute('data-id', docSnapshot.id);
    imageGridItem.setAttribute('data-author', data.author);
    imageGridItem.setAttribute('data-category', data.category);
    imageGridItem.setAttribute('data-description', data.description);

    const img = document.createElement('img');
    img.src = data.url;
    img.alt = data.description || 'Image';
    imageGridItem.appendChild(img);

    const button = document.createElement('button');
    button.className = 'toggle-context';
    button.textContent = '▼';
    button.onclick = () => openImageContextModal(docSnapshot.id);
    imageGridItem.appendChild(button);

    categoryContainer.appendChild(imageGridItem);
  }
}
```

Figure 3

```
<!-- Food Category -->
<section id="foodSection">
  <header>
    <h2 id="foodImagesHeader">Food</h2>
  </header>
  <!-- The container where dynamic content will be inserted -->
  <div class="food-images image-gallery">
    <!-- JavaScript will append image-grid-item elements here -->
  </div>
</section>
```

Figure 4

Sprint 2 – Recognising the importance of user experience, I prioritised the development of a navigation bar so they can navigate the page effortlessly on where they want to go (Michaela, 2022). A well-designed navigation bar is essential for seamless website navigation, guiding users effortlessly to their desired content. After researching, I adapted a navigation template, customising it to fit the project's needs (Figure 5, 6). The resulting navigation bar was both comprehensive and professional, aligning with my objectives for this sprint.

```
<div class="nav_bar">
  <div class="top_page_nav">
    <ul> <!-- List for navigation bar text -->
      <li><button id="aboutButton" class="nav-btn" data-target="aboutSection">About</button></li>
      <li><button id="foodImagesButton" class="nav-btn" data-target="foodSection">Food</button></li>
      <li><button id="fashionImagesButton" class="nav-btn" data-target="fashionSection">Fashion</button></li>
      <li><button id="sportsImagesButton" class="nav-btn" data-target="sportsSection">Sports</button></li>
      <li><button id="informativeImagesButton" class="nav-btn" data-target="informativeSection">Informative</button></li>
      <li><button id="funnyImagesButton" class="nav-btn" data-target="funnySection">Funny</button></li>
      <li><button id="historyImagesButton" class="nav-btn" data-target="historySection">History</button></li>
      <li><button id="otherImagesButton" class="nav-btn" data-target="otherSection">Other</button></li>
      <li><button id="signInButton" class="nav-btn">Log In</button></li>
      <div id="signInModal" class="modal">
        <div class="modal-content">
          <span class="close" id="closeSignInModalButton">&times;</span>
          <h3>Log In</h3>
          <form id="signInForm">
            <input type="email" id="signInEmail" placeholder="Email" required>
            <input type="password" id="signInPassword" placeholder="Password" required>
            <button type="submit" id="buttonSignIn">Log In</button>
          </form>
        </div>
      </div>
      <li><button id="signUpButton" class="nav-btn">Sign Up</button></li>
      <div id="signUpModal" class="modal">
        <div class="modal-content">
          <span class="close" id="closeSignUpModalButton">&times;</span>
          <h3>Sign Up</h3>
          <form id="signUpForm">
            <input type="email" id="signUpEmail" placeholder="Email" required>
            <input type="password" id="signUpPassword" placeholder="Password" required>
            <input type="password" id="signUpPasswordMatch" placeholder="Confirm Password" required>
            <button type="submit" id="buttonSignUp">Sign Up</button>
          </form>
        </div>
      </div>
      <li><button id="signOutButton" class="nav-btn">Log Out</button></li>
    </ul>
  </div>
  <!-- UPLOAD BUTTON -->
  <div id="bottom_page_nav">
    <ul>
      <li><button id="openModalButton">Upload Image</button></li>
    </ul>
  </div>
</div>
```

Figure 5

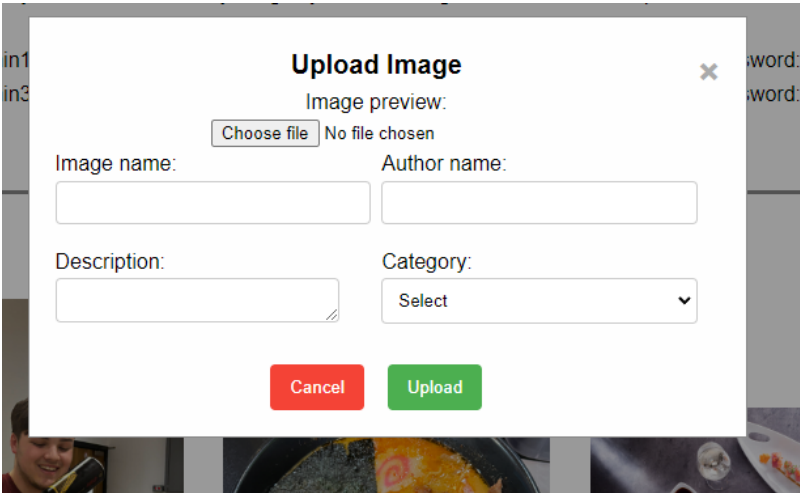
Echo Gallery



Figure 6

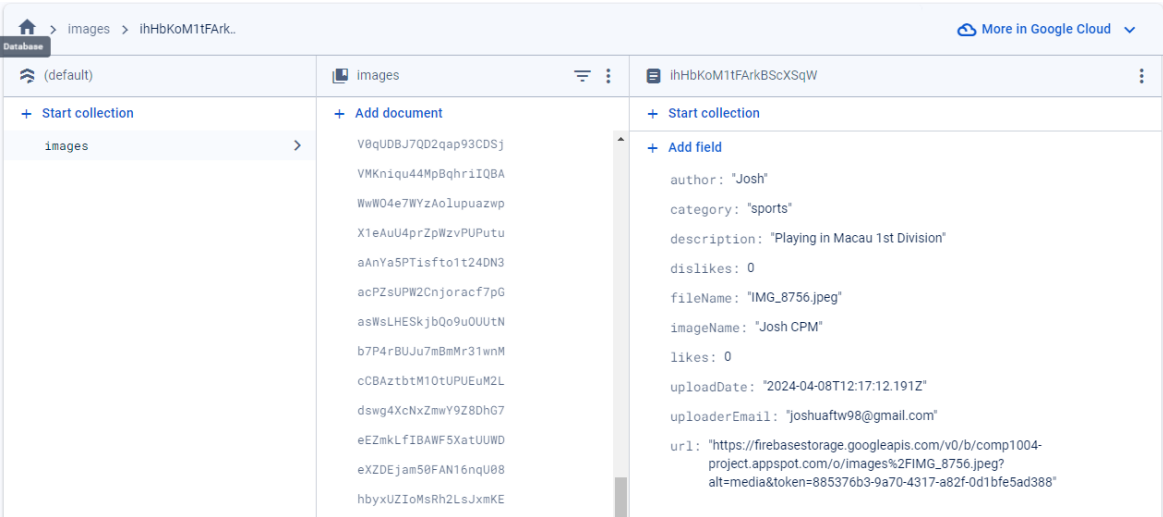
Sprint 3 – Developing the image upload functionality was crucial in my project, I implemented a Modal (W3Schools, 2019) that opens a user-friendly window for uploading images (Figure 7). The image upload function is within the DOM and retrieves all the data inputs from HTML. As stated previously, the use of Firestore database made things a lot easier, as I would use its capabilities for storing and retrieving data efficiently (Figure 8). By structuring the database to accommodate image metadata and relying on Firebase Storage

for file handling, I created a user-friendly and responsive image upload system. This process has not only simplified the management of image data but also enhanced the overall user experience by providing immediate and clear feedback throughout the upload process. The accompanying JavaScript code (Figure 9) orchestrates the image upload logic, interfacing with Firebase to store image details and retrieve the necessary data. Mastering CSS to ensure visual appeal was a significant part of this sprint's work.



The image shows a modal window titled "Upload Image" with a close button (X) in the top right corner. Inside the modal, there is an "Image preview:" label above a placeholder box that says "Choose file" and "No file chosen". Below this, there are four input fields: "Image name:" and "Author name:" are text inputs, "Description:" is a text input with a small icon at the bottom right, and "Category:" is a dropdown menu with "Select" and a downward arrow. At the bottom of the modal, there are two buttons: a red "Cancel" button and a green "Upload" button.

Figure 7



The image shows the Firebase Database console. The left sidebar shows the database structure with a collection named "images". The main area displays a document with the following fields:

Field	Value
author	"Josh"
category	"sports"
description	"Playing in Macau 1st Division"
dislikes	0
fileName	"IMG_8756.jpeg"
imageName	"Josh CPM"
likes	0
uploadDate	"2024-04-08T12:17:12.191Z"
uploaderEmail	"joshuaftw98@gmail.com"
url	"https://firebasestorage.googleapis.com/v0/b/project.appspot.com/o/images%2FIMG_8756.jpeg?alt=media&token=885376b3-9a70-4317-a82f-0d1bfe5ad388"

Figure 8

```

if (file) {
  try {
    const imageRef = storageRef(storage, `images/${file.name}`);
    const snapshot = await uploadBytes(imageRef, file);
    const url = await getDownloadURL(imageRef);

    const imagesCollectionRef = collection(db, 'images');
    const docRef = await addDoc(imagesCollectionRef, {
      fileName: file.name,
      imageName: imageName,
      category: category,
      description: description,
      author: author,
      uploaderEmail: uploaderEmail,
      likes: 0,
      dislikes: 0,
      uploadDate: currentDate.toISOString(),
      url: url
    });

    console.log('Document written with ID: ', docRef.id);
    alert('Image uploaded successfully!');

    document.getElementById('imageUploadForm').reset();

    document.getElementById('imagePreview').style.display = 'none';
    closeUploadModal();
    // Update Images on page
    await displayImagesByCategory(category);
  } catch (error) {
    console.error('Error during the upload:', error);
    alert('An error occurred during the upload.');
  }
} else {
  alert('Please select a file to upload.');
}

```

Figure 9

Sprint 4 – The sprint was smoother than anticipated. Enhancements to the modal from Sprint 3 involved adding fields for image name, author, description, and category (Figure 7), key for efficient content categorisation and retrieval. On upload, these details form a structured data object, sent to Firestore with asynchronous HTTP requests, ensuring metadata is searchable. These changes, requiring minor tweaks to the existing function due to robust groundwork, have improved the image gallery's organisation and accessibility, significantly boosting functionality and user engagement.

Sprint 5 – In retrospect, a week was more than needed for this sprint, thanks to DOM and Firebase integration. Using DOM manipulation, I dynamically rendered category-specific images from Firestore, with the categories array defined in the DOM (Figure 2), and the forEach loop invoking the 'displayImagesByCategory' asynchronous function (Figure 3), the project's responsiveness was significantly improved. Additionally, I implemented Firebase Authentication (Figure 10) to secure user sign-in and enable user-related features, like comments. This not only improved interactivity and security but also laid groundwork for advanced features, requiring users to sign in for full functionality. These developments were a leap forward for the project.

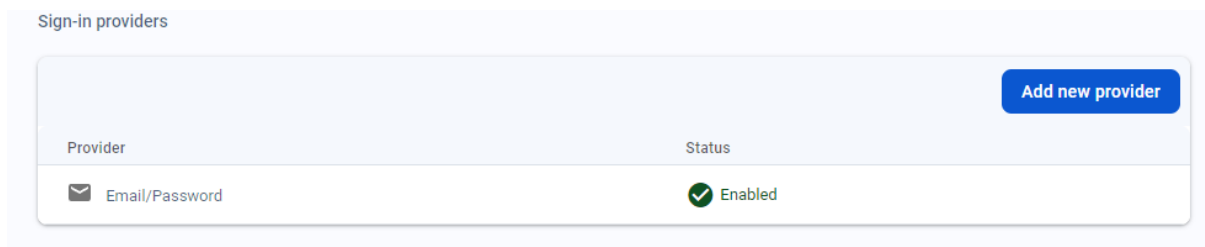


Figure 10

Sprint 6 – This sprint significantly boosted user engagement. I utilised JavaScript's template literals to generate interactive HTML elements like a down arrow, initiating an overlay modal that presents expanded image details, including the uploader's name and description (Figure 11). This method keeps users on the page by providing additional content through a floating overlay. The modal's dynamic HTML, sourced from Firebase, structures information in a user-centric way. This enhancement not only improved immediate access to image specifics but also set the stage for further interactive elements such as comments and reactions within the same modal context. The sprint was key to crafting an engaging and informative gallery experience.

```
// Load the content for the image
const dynamicContentHtml = `
<h3 class="image-detail image-imageName">${data.imageName}</h3>
<h4 class="image-detail image-category">${data.category}</h4>

<div style="display: flex; justify-content: space-between; align-items: center; margin-top: 10px;">
  <p><span class="static-text">Uploaded by: </span><span class="image-detail image-author">${data.author}</span></p>
  <p>Uploaded: ${uploadDate.toLocaleDateString()}</p>
</div>
<div style="display: flex; justify-content: space-between; margin-top: 10px;">
  <p><span class="static-text">Description: </span><span class="image-detail image-description">${data.description}</span></p>
  <div class="reaction-container">
    <button id="like-button-${docId}" class="reaction-button" aria-label="like">
      <span class="material-icons">thumb_up</span>
      <span id="likes-count-${docId}">${data.likes || 0}</span>
    </button>
    <button id="dislike-button-${docId}" class="reaction-button" aria-label="dislike">
      <span class="material-icons">thumb_down</span>
      <span id="dislikes-count-${docId}">${data.dislikes || 0}</span>
    </button>
  </div>
</div>
<div style="display: flex; justify-content: space-between;">
  <h5>Comments</h5>
  <div class="reaction-container">
    ${deleteButtonHtml}
    ${editButtonHtml}
  </div>
</div>
<div id="comments-container-${docId}" style="max-height: 150px; overflow-y: auto;">
<div style="display: flex; justify-content: space-between; align-items: center; margin-top: 10px;">
  <textarea id="comment-input-${docId}" placeholder="Add a comment..." style="width: 85%; height: 50px;">
  <button id="post-comment-button-${docId}" style="width: 100px;">Post Comment</button>
</div>
</div>
`;
```

Figure 11

Sprint 7 – This sprint was intricate, focusing on user interaction through a new commenting feature on images. A dynamic comment system was embedded into the image context modal, with a textbox for user input and a posting function for interactivity (Figure 11). Additionally, I integrated like and dislike buttons with real-time counters, relying on Firestore's "comments" and "reactions" subcollections (Figure 12). Subcollection initialisation and features are triggered by user actions—likes, dislikes, or comment posts (Figure 13, Figure 14). Commenting involved saving user details and timestamps, with display functionality bringing comments into the modal (Figure 15). I initialised likes and dislikes at zero upon image upload to ensure accurate interaction tracking.

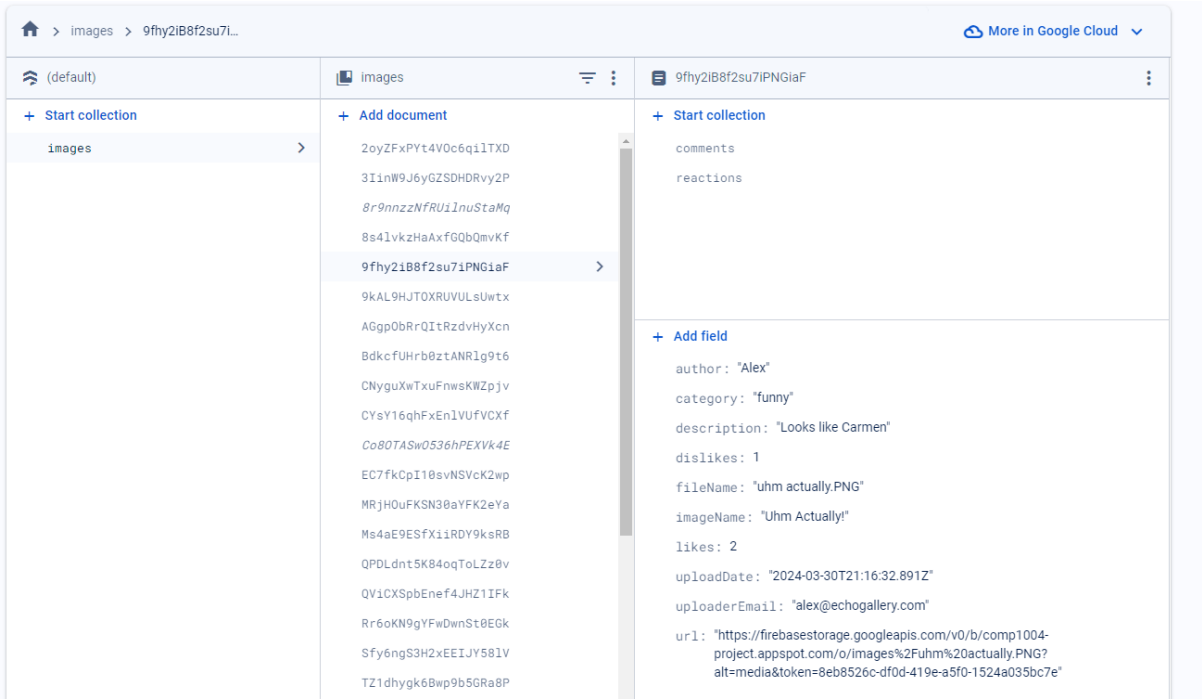


Figure 12

```

// Submit Comment function
async function submitComment(docId) {
  const commentInput = document.getElementById(`comment-input-${docId}`);
  const commentText = commentInput.value.trim();
  const auth = getAuth();
  const user = auth.currentUser;

  // Set a maximum word limit
  const maxWordLimit = 100;
  const wordCount = commentText.split(/\s+/).length;

  if (commentText === '') {
    alert('Comment cannot be empty.');
```

```

    return;
  }

  if (!user) {
    alert('You must be logged in to post comments.');
```

```

    return;
  }

  if (wordCount > maxWordLimit) {
    alert(`Comment cannot exceed ${maxWordLimit} words. You have used ${wordCount} words.`);
    return;
  }

  try {
    const commentsRef = collection(db, `images/${docId}/comments`);
    await addDoc(commentsRef, {
      text: commentText,
      user: user.email,
      timestamp: serverTimestamp() // Firebase server timestamp for comment ordering
    });

    // Clear the comment input field
    commentInput.value = '';

    // Refresh the comments section to include the new comment
    await displayComments(docId);
  } catch (error) {
    console.error('Error posting comment:', error);
    alert('Failed to post comment.');
```

```

  }
}

```

Figure 13

```

async function updateLikes(docId, userId, isLike) {
  const imageDocRef = doc(db, 'images', docId);
  const userReactionRef = doc(db, `images/${docId}/reactions`, userId);

```

Figure 14

```

// Display Comment Function
async function displayComments(docId) {
  const commentsContainer = document.getElementById(`comments-container-${docId}`);
  const commentsRef = collection(db, `images/${docId}/comments`);
  const q = query(commentsRef, orderBy("timestamp", "asc")); // Order by timestamp
  const querySnapshot = await getDocs(q);

  // Clear previous comments
  commentsContainer.innerHTML = '';

  querySnapshot.forEach((doc) => {
    const commentData = doc.data();
    const commentElement = document.createElement('p');
    commentElement.textContent = `${commentData.user}: ${commentData.text}`;
    commentsContainer.appendChild(commentElement);
  });
}

```

Figure 15

Sprint 8 – In this sprint I created edit and delete functions (Figure 16, 17) that are only accessible to the user that uploaded the image (Figure 18). I made it so that these buttons would only appear if the uploaded user email matched with the logged in user's email address (Figure 19, 20). I thought these would be important as previously I was deleting images through Firestore. It also meant that users were able to edit the descriptions, categories, or uploader names of any images that they have uploaded, creating a more professional project. This sprint presented the most difficulties, which are noted in 'Issues and Constraints' section of this report.

```

// Toggle between edit and view mode
function toggleEdit(docId, buttonElement) {
  const isEditing = buttonElement.textContent === 'Edit Details';
  const detailsFields = ['imageName', 'category', 'description', 'author'];

  // If currently editing, switch to save mode
  if (isEditing) {
    buttonElement.textContent = 'Save Changes';

    // Store the old category in the dataset of the button
    const categoryElement = document.querySelector(`#imageContextContent .image-category`);
    buttonElement.dataset.oldCategory = categoryElement.textContent.trim();

    // Convert each detail field into an editable input or select
    detailsFields.forEach(field => {
      const fieldElement = document.querySelector(`.image-${field}`);
      if (fieldElement) {
        if (field === 'category') {
          // Select element for category
          const selectElement = document.createElement('select');
          selectElement.innerHTML = `
            <option value="food">Food</option>
            <option value="fashion">Fashion</option>
            <option value="sports">Sports</option>
            <option value="informative">Informative</option>
            <option value="funny">Funny</option>
            <option value="history">History</option>
            <option value="other">Other</option>
          `;
          selectElement.value = fieldElement.textContent;
          selectElement.classList.add('image-detail-input');
          selectElement.dataset.field = field;
          fieldElement.textContent = '';
          fieldElement.appendChild(selectElement);
        } else {
          // Create an input element for other fields
          const inputElement = document.createElement('input');
          inputElement.type = 'text';
          inputElement.value = fieldElement.textContent;
          inputElement.classList.add('image-detail-input');
          inputElement.dataset.field = field;
          fieldElement.textContent = '';
          fieldElement.appendChild(inputElement);
        }
      } else {
        console.error(`Element with class .image-${field} not found.`);
      }
    });
  } else {

```

Figure 16

```

// Function to delete a post and subcollection
async function deletePost(docId) {
  const confirmation = confirm('Are you sure you want to delete this post?');

  if (!confirmation) {
    return;
  }

  // Start a batch operation as we have image and subcollection comments
  const batch = writeBatch(db);

  // Reference to the image document
  const imageDocRef = doc(db, 'images', docId);
  batch.delete(imageDocRef);

  // Reference to the comments subcollection of the image
  const commentsCollectionRef = collection(db, `images/${docId}/comments`);
  const reactionsCollectionRef = collection(db, `images/${docId}/reactions`);

  try {
    // Get all comments and reactions associated with the image
    const commentsSnapshot = await getDocs(commentsCollectionRef);
    commentsSnapshot.forEach((commentDoc) => {
      batch.delete(commentDoc.ref);
    });

    const reactionsSnapshot = await getDocs(reactionsCollectionRef);
    reactionsSnapshot.forEach((reactionDoc) => {
      batch.delete(reactionDoc.ref);
    });

    // Commit the batch to delete the image and all comments
    await batch.commit();
    alert('The post has been deleted.');
```

closeImageContextModal();

```

    const imageElement = document.querySelector(`[data-id="${docId}"]`);
    if (imageElement) {
      imageElement.remove();
    }
  } catch (error) {
    console.error('Error deleting post and subcollection:', error);
    alert('An error occurred while trying to delete the post.');
```

}

```


}

```

Figure 17

Vegan Cupcakes


Food




Uploaded by: Alex

Uploaded: 30/03/2024

Description: Not enough frosting

 0

 0

Comments

Post

Comment

Figure 18

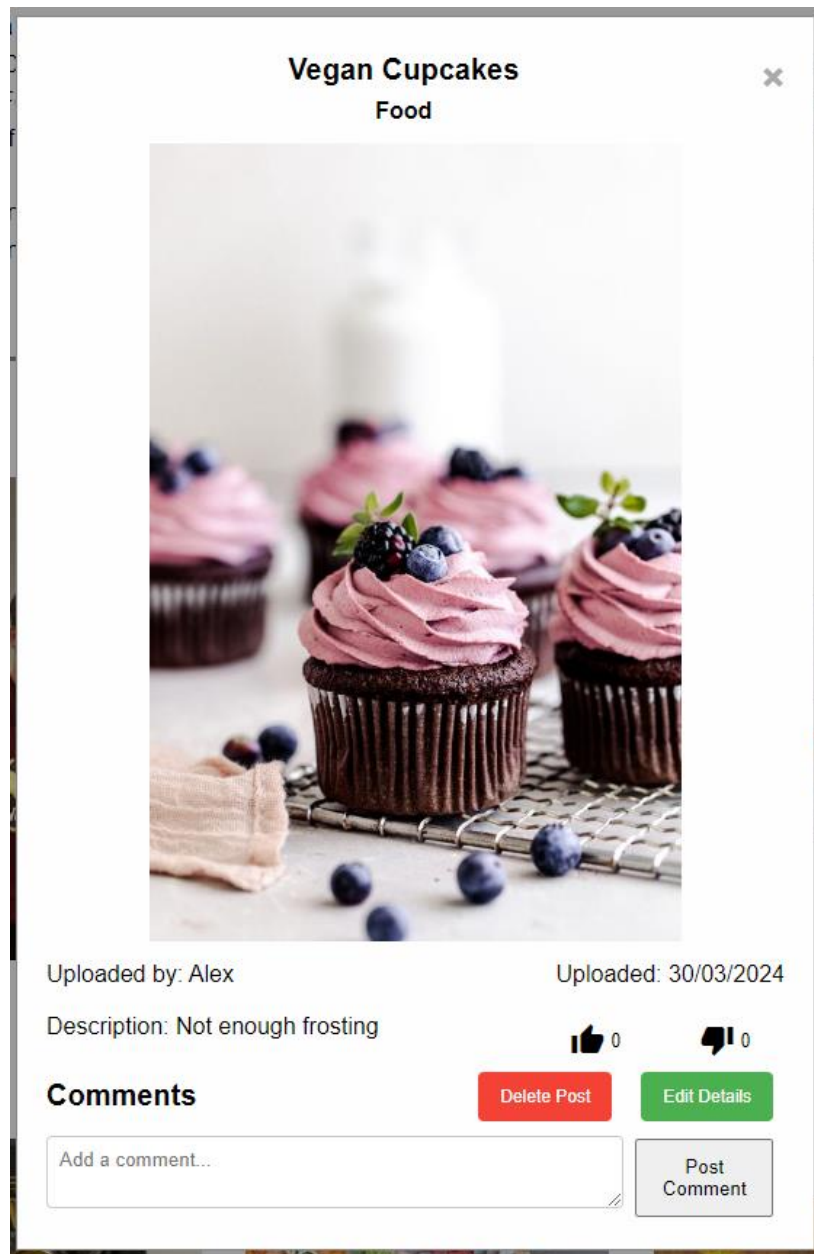


Figure 19

```
const auth = getAuth();
const user = auth.currentUser;

// If user is logged in show these buttons
let deleteButtonHtml = '';
if (user && data.uploaderEmail === user.email) {
  deleteButtonHtml = `<button onclick="deletePost('${docId}')" class="delete-button">Delete Post</button>`;
}

let editButtonHtml = '';
if (user && data.uploaderEmail === user.email) {
  editButtonHtml = `<button id="editButton-${docId}" class="edit-button">Edit Details</button>`;
}
```

Figure 20

Sprint 9 – The final stage was crucial; I set up a free domain (<https://comp1004-azure.vercel.app/>) for friends to test and upload images. A footer with a Google feedback form was added for anonymous suggestions and bug reports (Figure 20, Figure 21). Feedback highlighted a bug preventing 'unliking' or 'undisliking' images, which I quickly fixed with additional conditional logic. This refinement led to the completion of my report.

The image shows a screenshot of a Google Feedback form. It is divided into two sections. The first section is titled 'What did you like about the website?' and shows 7 responses. The second section is titled 'Any other feedback or improvements you would like to see?' and shows 6 responses. The responses are listed in a scrollable area below each question.

What did you like about the website?
7 responses

- responsive and fun to use
- It's similar to Pinterest for sharing ideas/trends through images
- I like that you can upload your own photos for everyone to see and comment on
- It is engaging and fun to use with friends and family
- Simple enough, straight forward. Easy enough to navigate to an area to wanna look at, as well doom scroll through all the sections.
- The UI is intuitive and the functionality is great
- It has a no thrills, simple to use UI. I also like the randomisation of the images.

Any other feedback or improvements you would like to see?
6 responses

- improvement to UI, functionality is great
- Maybe add some more categories, for example animals/travel/landscapes
- personalised usernames
- it was cool that I could move my upload from one category to another one
- If you were thinking about mobile usage (data usage), having all the images viewable immediate surely will eat a lot of memory up front, so maybe consider separate pages to cut down on usage.
- Well done Alex, this is really good!

Figure 20

Website Form

COMP1004 Project Feedback

alex.t.draper@gmail.com [Switch account](#)

Not shared

* Indicates required question

I found the user interface easy to navigate and professional *

1

2

3

4

5

Strongly Disagree

☐

☐

☐

☐

☐

Strongly Agree

The website was easy to understand and function *

1

2

3

4

5

Strongly Disagree

☐

☐

☐

☐

☐

Strongly Agree

The terminology was clear and precise *

1

2

3

4

5

Strongly Disagree

☐

☐

☐

☐

☐

Strongly Agree

Were you able to see other peoples images and view the details about them by clicking the down arrow? *

☐ No

☐ Yes

Were you able to log in and/or create an account? *

☐ No

☐ Yes

Figure 21

Firestore Implementation Overview

Firestore has played a significant role in this project, providing a range of backend services that are efficient and easy to implement. Its real-time database ensures instant data sync, creating a responsive user experience. Firestore's authentication services streamline secure session management, while its storage options offer reliable file hosting. The platform's analytics offer critical insights for continuous improvement. Overall, Firestore has accelerated development, reduced backend upkeep, and allowed a greater focus on frontend development, aiding in the project's timely completion.

Sprint Summary

Throughout the implementation, I adhered to advanced software engineering principles, evolving the project from concept to a user-centric platform. The project evolved from a basic concept into a platform focused on user needs, highlighting the importance of adaptability and continuous user engagement. Agile methodologies steered the process, aligning with strategic objectives and enabling me to adeptly navigate challenges. The codebase, embodying engineering rigour with principles like DRY, YAGNI, KISS, and SOLID, culminated in a feature set that aligned seamlessly with project goals.

Testing

Test Case	Objective	Steps	Expected Result	Pass/Fail
Display Images by Category	Check if images are correctly categorised and displayed.	1. Select a category from the navigation menu. 2. Observe the displayed images.	Only images from the selected category are displayed.	Pass
Sign Up Functionality	Ensure users can register using their email and password.	1. Navigate to the sign-up modal. 2. Enter valid email and matching passwords. 3. Submit the form.	User is registered, and a welcome message is displayed.	Pass
Sign In Functionality	Ensure registered users can sign in.	1. Navigate to the sign-in modal. 2. Enter registered email and correct password. 3. Submit the form.	User is logged in, and the interface updates accordingly.	Pass
Sign Out Functionality	Ensure users can securely sign out.	1. Click on the 'Sign Out' button.	User is logged out; UI reverts to non-authenticated state.	Pass
Image Upload	Validate that users can upload images correctly.	1. Ensure user is logged in. 2. Navigate to the upload modal. 3. Select an image file and fill out all required fields. 4. Submit the upload form.	Image is uploaded, stored, and displayed appropriately.	Pass
Like and Dislike Updates	Ensure likes and dislikes are recorded and updated in real-time.	1. Click the like or dislike button on any image. 2. Observe the counters next to the buttons.	The counters update immediately reflecting the user's action.	Pass
Post Comments	Validate that users can leave comments on images.	1. Open an image modal. 2. Enter a comment in the comment box. 3. Submit the comment.	Comment appears under the image instantly.	Pass
Navigation Bar Usability	Test the effectiveness of the navigation bar.	1. Use the navigation bar to move between sections.	Each button leads to the correct section.	Pass
Open Image Context Modal Functionality	Ensure that the modal with image details opens correctly and displays the appropriate information.	1. Navigate to a category and view the images. 2. Click the ▼ button on any image grid item. 3. Observe the modal that opens.	The modal displays detailed information about the image including likes, comments, and description.	Pass
Edit and Delete Button Visibility	Verify that the edit and delete button is visible only to users who uploaded the image.	1. Sign in with a user who uploaded an image. 2. Open the context modal for the uploaded image. 3. Check for the presence of the edit and delete button.	The edit and delete button are visible only to the uploader.	Pass
Delete Button Functionality	Check that the delete button functions correctly and removes the image from the database and UI.	1. Sign in and navigate to an image uploaded by the user. 2. Open the context modal. 3. Click the delete button and confirm deletion.	The image is deleted from Firebase and the UI updates to reflect the change.	Pass
Edit Functionality	Ensure that the edit button allows for modifying image details like name, category, and description.	1. Open the context modal for an image uploaded by the user. 2. Click the edit button. 3. Change the image details. 4. Submit changes.	The image details are updated in the database and the UI reflects these changes immediately.	Pass
Validation of Edit Inputs	Check that the input validation works correctly during the edit process (e.g., no empty fields, valid categories).	1. Trigger the edit mode. 2. Clear the description field. 3. Attempt to save changes. 4. Try changing the category to an invalid value and save.	Appropriate error messages are displayed, and no changes are saved to the database for invalid inputs.	Pass
Persistence of Edited Details	Confirm that changes made to image details persist across sessions and are visible to all users.	1. Edit an image and submit changes. 2. Log out and then log back in. 3. View the edited image and check details. 4. Have another user log in and check the details.	Edited details are consistent and visible to all users, confirming changes are persisted correctly.	Pass
Image Details Accuracy	Ensure that the details displayed in the modal are accurate and match those stored in the database.	1. Select an image. 2. Click the ▼ button to open the modal. 3. Verify displayed details against expected details	All details in the modal match the database records.	Pass

Issues and Constraints

Sprint 8 posed challenges with image re-categorisation and deletion, emphasising the complexity of client-side rendering alongside database updates. Users encountered a bug when changing an image's category; the image would duplicate, appearing in both the old and new categories. This underscored the necessity of thorough testing and iterative development.

The issue persisted, demanding a nuanced understanding of DOM updates in reaction to data alterations. I revised the image rendering code, introducing a check to ensure an image only appeared in its assigned category. This was a learning curve, involving debugging and grasping the asynchronous nature of JavaScript and Firestore interactions. The solution entailed updating the image display logic to remove the image from the old category before re-rendering it in the new one (Figure 23). This fix not only resolved the issue but also served as a valuable lesson in the intricacies of web application state management.

```
// Firestore update on edit
async function updateImageDetails(docId, updatedData, oldCategory) {
  const imageDocRef = doc(db, 'images', docId);

  try {
    await updateDoc(imageDocRef, updatedData);
    alert('Image details updated successfully.');
```

```
    // Check if category has changed and remove the image from the old category
    if (oldCategory !== updatedData.category) {
      const oldCategoryContainer = document.querySelector(`.${oldCategory}-images`);
      const imageElement = oldCategoryContainer ? oldCategoryContainer.querySelector(`[data-id="${docId}"]`) : null;
      if (imageElement) {
        oldCategoryContainer.removeChild(imageElement);
      }
      await displayImagesByCategory(updatedData.category);
    }
    closeImageContextModal();
  } catch (error) {
    console.error('Error updating image details:', error);
    alert('An error occurred while updating the details. Please try again.');
```

```
  }
}
```

Figure 23

Addressing the next challenge, I found that deleting images from Firestore didn't automatically remove their associated comments and reactions subcollections. This shed light on the importance of handling related data and the use of atomic transactions in database operations.

To solve this, I developed a deletion process that executed a batch operation, ensuring the simultaneous removal of the image document and its related comments and reactions (Figure 24). This approach was critical for thorough data removal, preventing orphaned records and preserving the integrity of the application's data structure.

```

// Function to delete a post and subcollection
async function deletePost(docId) {
  const confirmation = confirm('Are you sure you want to delete this post?');

  if (!confirmation) {
    return;
  }

  // Start a batch operation as we have image and subcollection comments
  const batch = writeBatch(db);

  // Reference to the image document
  const imageDocRef = doc(db, 'images', docId);
  batch.delete(imageDocRef);

  // Reference to the comments subcollection of the image
  const commentsCollectionRef = collection(db, `images/${docId}/comments`);
  const reactionsCollectionRef = collection(db, `images/${docId}/reactions`);

  try {
    // Get all comments and reactions associated with the image
    const commentsSnapshot = await getDocs(commentsCollectionRef);
    commentsSnapshot.forEach((commentDoc) => {
      batch.delete(commentDoc.ref);
    });

    const reactionsSnapshot = await getDocs(reactionsCollectionRef);
    reactionsSnapshot.forEach((reactionDoc) => {
      batch.delete(reactionDoc.ref);
    });

    // Commit the batch to delete the image and all comments
    await batch.commit();
    alert('The post has been deleted.');
```

closeImageContextModal();

```

    const imageElement = document.querySelector(`[data-id="${docId}"]`);
    if (imageElement) {
      imageElement.remove();
    }
  } catch (error) {
    console.error('Error deleting post and subcollection:', error);
    alert('An error occurred while trying to delete the post.');
```

}

```

}

```

Figure 24

Managing the project as an SPA was both challenging and restrictive, necessitating a calculated approach to design and user experience. SPAs demand meticulous state management, dynamic content loading, and performance optimisation to ensure a smooth and responsive interface.

One of the primary challenges was to ensure that the entire user experience happened within a single page, without the traditional page reloads associated with multi-page

applications. This meant that all interactions, from image uploads, edits, category changes, to authentication flows, had to be handled dynamically. JavaScript played a pivotal role in this, providing the necessary tools to update the DOM in real time, reflecting changes immediately to the user without a full-page refresh.

Despite its complexities, the SPA framework offers benefits like enhanced speed and seamless user experiences, as it refreshes only the needed content. It also complements contemporary web development practices, delivering an app-like experience users expect.

Adhering to SPA principles, I successfully incorporated all functionalities, ensuring the app remained responsive and user-friendly. Leveraging Firebase's real-time data updates and authentication services meshed well with the SPA structure. The outcome was a unified, efficient, and approachable application that fulfilled project goals while providing an engaging user experience.

Reflection

Reflecting on the project, the progression from idea to execution has been immensely fulfilling. The primary objective was to craft an SPA for image sharing with comprehensive features like image management and user interactions, supported by Firebase's robust backend capabilities, which have been instrumental in the application's success.

The achievement of a dynamic user interface stands out, allowing seamless user engagement. The SPA challenges were addressed through innovative methods like asynchronous JavaScript and Firebase's instantaneous database updates.

However, there are areas of potential growth and additional features that could be implemented:

- **Comment Management:** Allowing users to delete their comments will enable better control over their content, fostering a more secure and comfortable environment.
- **Custom Usernames:** Personalised usernames could create a more community-focused experience.
- **Like Transparency:** Revealing who has liked an image can encourage more social interaction and connection among users, giving insights into common interests.
- **Administrative Control:** Implementing administrative rights to delete any image would be crucial for content moderation and maintaining community standards.
- **Following Mechanism:** Adding the ability to follow users can create a more engaging social platform, encouraging regular visits and interactions.
- **Private Profiles:** Providing privacy settings would give users control over their content visibility.
- **Bulk Image Uploads:** A multiple image upload feature can streamline the sharing process, making the platform more user-friendly for those with multiple images to share.
- **Expand Categories:** Introduce more categories in the navigation bar, such as art, 'most liked' to enhance user experience.

These features would not only enhance user engagement but also contribute to a safer and more interactive community. They require careful consideration of user privacy and data security, ensuring compliance with regulations and ethical standards.

On a personal note, this project has been a learning curve, especially in terms of managing user experience and state within an SPA. The platform's interactivity highlighted the necessity of crafting a user-friendly interface and the importance of adept data management.

The iterative development process has been enriched by integrating user feedback, providing valuable insights into desired future enhancements.

In conclusion, the project exemplifies the strengths of contemporary web development frameworks and the expansive potential of cloud services. This project not only served as a practical application of modern web development principles but also as an educational journey through the SDLC. It underlines the framework's critical role in project success and provides a blueprint for future endeavours. The lessons learnt from this experience will undoubtedly inform and enhance future projects.

GitHub repo link:

<https://github.com/Alex-T-Draper/Comp1004/tree/main/Project>

References:

Chaffey, D. (2024). *Global Social Media Research Summary 2024*. [online] Smart Insights. Available at: <https://www.smartinsights.com/social-media-marketing/social-media-strategy/new-global-social-media-research/> [Accessed 31 Mar. 2024].

Development, L.S. (2023). *Significance of Software Development Life Cycle (SDLC)*. [online] Medium. Available at: <https://leeddev.medium.com/significance-of-software-development-life-cycle-sdlc-3617338d3883> [Accessed 31 Mar. 2024].

GDPR (2018). *General data protection regulation (GDPR)*. [online] General Data Protection Regulation (GDPR). Available at: <https://gdpr-info.eu/> [Accessed 10 Apr. 2024].

Michaela (2022). *Why is website navigation so important?* [online] blog.wurkhouse.com. Available at: <https://blog.wurkhouse.com/importance-website-navigation-best-practices> [Accessed 6 Apr. 2024].

Ortiz-Ospina, E. (2019). *The Rise of Social Media*. [online] Our World in Data. Available at: <https://ourworldindata.org/rise-of-social-media> [Accessed 31 Mar. 2024].

Pinheiro, J. (2018). *Software Development Life Cycle (SDLC) phases*. [online] Medium. Available at: <https://medium.com/@jilvanpinheiro/software-development-life-cycle-sdlc-phases-40d46afbe384> [Accessed 31 Mar. 2024].

University of Cumbria (2019). *Why use social media? | MyCumbria*. [online] Cumbria.ac.uk. Available at: <https://my.cumbria.ac.uk/Student-Life/it-media/Social-Media-Guidance/Why-use-social-media/> [Accessed 31 Mar. 2024].

W3C (2018). *Web Content Accessibility Guidelines (WCAG) Overview*. [online] Web Accessibility Initiative (WAI). Available at: <https://www.w3.org/WAI/standards-guidelines/wcag/> [Accessed 10 Apr. 2024].

W3Schools (2019). *How To Make a Modal Box With CSS and JavaScript*. [online] W3schools.com. Available at: https://www.w3schools.com/howto/howto_css_modals.asp [Accessed 6 Apr. 2024].